AF / 2183 / IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

In re Patent Application of

NEVILL et al.

Serial No. 09/887,559

Filed: June 25, 2001

Title: RESTARTING TRANSLATED INSTRUCTIONS

Atty Dkt. 550-242
C# M#
TC/A.U.: 2183
Examiner: O'Brien, B.
Date: May 10, 2005

OIPE JC25 MAY 1 0 2005 PATENT & TRADEMARK OFFICE

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

☐ **Correspondence Address Indication Form Attached.**

☐ **NOTICE OF APPEAL**
Applicant hereby **appeals** to the Board of Patent Appeals and Interferences from the last decision of the Examiner twice/finally rejecting applicant's claim(s). $500.00 (1401)/$250.00 (2401) $

☒ An appeal **BRIEF** is attached in the pending appeal of the above-identified application $500.00 (1402)/$250.00 (2402) $ 500.00

☐ Credit for fees paid in prior appeal without decision on merits -$ ( )

☐ A reply brief is attached. **(no fee)**

☐ Petition is hereby made to extend the current due date so as to cover the filing date of this paper and attachment(s)
One Month Extension $120.00 (1251)/$60.00 (2251)
Two Month Extensions $450.00 (1252)/$225.00 (2252)
Three Month Extensions $1020.00 (1253/$510.00 (2253)
Four Month Extensions $1590.00 (1254/$795.00 (2254) $

☐ "Small entity" statement attached.

Less month extension previously paid on -$( )

TOTAL FEE ENCLOSED $ 0.00

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension. The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this firm) to our **Account No. 14-1140.** A duplicate copy of this sheet is attached.

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:sd

NIXON & VANDERHYE P.C.
By Atty: John R. Lastova, Reg. No. 33,149

Signature: John R. Lastova

953440

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

NEVILL et al. Atty. Ref.: 550-242

Serial No. 09/887,559 Group: 2183

Filed: June 25, 2001 Examiner: O'Brien, B.

For: RESTARTING TRANSLATED INSTRUCTIONS

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT
On Appeal From Final Rejection
From Group Art Unit 2183

John R. Lastova
NIXON & VANDERHYE P.C.
8th Floor, 1100 North Glebe Road
Arlington, Virginia 22201-4714
(703) 816-4025
Attorney for Appellant
Nevill et al.
ARM Limited

TABLE OF CONTENTS

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

NEVILL et al. — Atty. Ref.: 550-242

Serial No. 09/887,559 — Group: 2183

Filed: June 25, 2001 — Examiner: O'Brien, B.

For: RESTARTING TRANSLATED INSTRUCTIONS

* *

May 10, 2005

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Sir:

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a corporation of the United Kingdom.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals related to this subject application. There are no interferences related to this subject application.

III. STATUS OF CLAIMS

Claims 1-17 are pending. Claims 1-7, 11-12, and 16-17 stand rejected under 35 U.S.C. §102 as being anticipated by USP 5,307,504 to Robinson. Claims 8-10 and 13-15 stand rejected under 35 U.S.C. §103 as being unpatentable based on Robinson combined with USP 5,898,885 to Dickol.

IV. STATUS OF AMENDMENTS

There have been no after final submissions.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Processor cores execute instructions from a "native" instruction set, and instruction translators may be used to translate instructions from a non-native instruction set into a native instruction format so they can be executed by the processor core. But a problem with instruction translation is how to deal with interrupt signals. In general, a processing system should respond to interrupt signals as rapidly as possible, particularly in systems controlling real time operations. Indeed, interrupt latency can be a critical performance parameter and is measured using a "worst case" interrupt response situation. The desirability of achieving low interrupt latency when executing non-native instructions is highlighted if one considers that one may wish to use such systems in real time applications, such as airbag control systems or anti-lock brake systems, in which the worst case interrupt latency may be a safety critical parameter.

To minimize interrupt latency, an interrupt is often responded to right after execution of a currently-executing native instruction. But in situations where a single non-native instruction is translated into more than one native instruction, there is a problem when an interrupt is received during the execution of a sequence of native instructions representing a single non-native instruction. In that case, the non-native instruction may be only partly have been completed, and the state of the processing system may be uncertain.

One way of dealing with this would be to provide additional hardware that was triggered upon receipt of an interrupt signal to store the current state of the processing system such that the state could be restored prior to restarting after the interrupt and so any partially completed non-native instruction would be able to be carried forward to completion. However, such an approach has the disadvantage of incurring an additional hardware overhead, significant additional complexity which may in turn degrade interrupt performance due to the need to save the state of the processing system prior to servicing the interrupt. An alternative approach would be to control the system such that non-native instructions are treated as "atomic", i.e., an interrupt is not serviced until after a non-native instruction fully completes its execution. This approach also adversely impacts interrupt latency.

A better solution provided by the inventors translates non-native instructions into a form that may take the equivalent of several native instructions to execute and provides interrupt servicing after completion of an operation corresponding to a native instruction without introducing undue interrupt latency or other difficulty/complexity on restarting.

This is achieved by arranging that the translated sequence of operations so that no changes are made to the input variables for that operation until the final native instruction is executed. Accordingly, if the interrupt occurs prior to the execution of the final operation, then execution of a non-native instruction being translated is restarted in its entirety because the input variables will be unaltered. On the other hand, if the interrupt occurs after starting the execution of the final operation, then the final operation completes, and the restart logic carries on from the next instruction following the non-native instruction during which the interrupt occurred.

Although the invention is applicable to many different types of instruction set, it is particularly useful when the non-native instructions specify operations to be executed upon stack operands held in a stack, e.g., Java bytecodes. Such stack-based systems typically read their input operands from the stack and write their output operands to the stack. When emulating such operation, stack operands are not overwritten until after execution of the final operation has commenced. Similarly, stack operands are not added to the stack until execution of the final operation has commenced.

Non-limiting example embodiments described are directed to Java bytecode translation examples. Figure 1 shows a first example instruction pipeline 30 of a type suitable for use in an ARM processor based system (Figure 2 shows another). The instruction pipeline 30 includes a fetch stage 32, a native instruction (ARM/Thumb instructions) decode stage 34, an execute stage 36, a memory access stage 38 and a write back stage 40. The execute stage 36, the memory access stage 38 and the write back stage 40 are substantially conventional. Downstream of the fetch stage 32 and upstream

of the native instruction decode stage 34 is an instruction translator stage 42. The instruction translator stage 42 is a finite state machine that translates non-native Java bytecode instructions of a variable length into native ARM instructions. The instruction translator stage 42 is capable of multi-step operation where a single Java bytecode instruction may generate a sequence of ARM instructions that are fed along the remainder of the instruction pipeline 30 to perform the operation specified by the Java bytecode instruction. Figure 3 illustrates the fetch stage of an instruction pipeline in more detail. Fetching logic 60 fetches instruction words from a memory system and supplies these to an instruction word buffer 62. An instruction translator 64 performs translation of non-native Java bytecodes into native ARM instructions.

Figure 5 shows a data processing system 102 including a processor core 104 and a register bank 106. An instruction translator 108 is provided within the instruction path to translate Java Virtual Machine instructions to native ARM instructions (or control signals corresponding thereto) that may then be supplied to the processor core 104. The register bank 106 in this example contains sixteen general purpose 32-bit registers, of which four are allocated for use in storing stack operands, i.e., the set of registers for storing stack operands is registers R0, R1, R2 and R3.

A convenient way of keeping track of how the system should be restarted if an interrupt does occur is to store a pointer to a restart location with the pointer being advanced upon execution of the final operation. This pointer may conveniently be a program counter (PC) value pointing to a memory address of a memory location storing an instruction currently being translated. A program counter value is associated with

each Java bytecode currently being translated. This program counter value is passed along the stages of the pipeline such that each stage is able, if necessary, to use the information regarding the particular Java bytecode it is processing. The program counter value for a Java bytecode that translates into a sequence of a plurality of ARM instruction operations is not incremented until the final ARM instruction operation within that sequence starts to be executed.

Figure 8 is a flow diagram schematically illustrating one non-limiting example set of detailed procedures for performing instruction translation and interrupt processing in the system of Figure 5. At step 10, a Java bytecode is fetched from memory. At step 12, "require full" (indicating the number of stack operands that must be present within the set of registers prior to the Java bytecode execution) and "require empty" (indicating the number of empty registers within the set of registers that must be available prior to execution of the ARM instructions representing the Java opcode) values for that Java bytecode are examined. If either of the require empty or require full conditions are not met, then respective PUSH and POP operations of stack operands (possibly multiple stack operands) may be performed in steps 14 and 16.

At step 18, the first ARM instruction specified within a translation template for the Java bytecode concerned is selected. Step 20 checks whether the selected ARM instruction is the final instruction to be executed in the emulation of the Java bytecode fetched at step 10. If so, then the program counter value is updated in step 21 to point to the next Java bytecode in the sequence of instructions to be executed. Because this is the final ARM instruction, its execution is completed irrespective of whether an interrupt

now occurs. Accordingly, it is safe to update the program counter value to the next Java bytecode and restart execution from that point because the state of the system will have reached that matching normal, uninterrupted, full execution of the Java bytecode. If the test at step 20 indicates that the final bytecode has not been reached, then updating of the program counter value is bypassed.

Step 22 executes the current ARM instruction. At step 24, a test is made as to whether there are any more ARM instructions requiring execution. If so, then the next is selected at step 26 and processing returns to step 20. If not, then processing proceeds to step 28 to map any change/swap specified for the Java bytecode concerned in order to reflect the desired top of stack location and full/empty status of the various registers holding stack operands.

Figure 8 also schematically illustrates the points at which an interrupt if asserted is serviced and then processing restarted after an interrupt. An interrupt starts to be serviced after the execution of an ARM instruction currently in progress at step 22 with whatever is the current program counter value being stored as a return point with the bytecode sequence. If the current ARM instruction executing is the final instruction within the translation template sequence, then step 21 will have just updated the program counter value. Accordingly, the PC value points to the next Java bytecode (or ARM instruction should an instruction set switch have just been initiated). If the currently executing ARM instruction is anything other than the final instruction in the sequence, then the program counter value stays the same as that indicated at the start of the

execution of the Java bytecode concerned. Consequently, when a return is made, the whole Java bytecode will be re-executed.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The primary rejection to be reviewed on appeal is the obviousness rejection based on Robinson et al., U.S. patent no. 5,307,504. The secondary rejection to be reviewed is that of claims 8-10 and 13-15 based on Robinson in view of Dickol et al., U.S. patent no. 5,898,885.

VII. ARGUMENT

A. The Robinson Reference

Robinson describes translating a complex instruction set code (CISC) to produce a program of reduced instruction set code (RISC). Each CISC instruction is translated into a sequence of RISC instructions, and each sequence includes four groups of instructions. Robinson seeks to "preserve instruction granularity in the translation process." Column 4, lines 18 and 19. The idea is that when several RISC instructions corresponding to the translated CISC instructions are being executed, they should produce the same result that the corresponding CISC instruction would have produced, even though "asynchronous events may occur during execution of any of the 'granules' of simpler translated instructions." Column 4, lines 24-26.

Robinson's focus is thus on preserving instruction granularity. See col.4, lines 11-23: "in translating CISC to RISC it is essential that instruction wholeness or granularity

be preserved...[A]ssurance must be provided that each set, or "grounds", of translated instructions corresponding to each more complex instruction will execute to produce the same result that the corresponding more complex instruction would have produced." This objective is different from improving interrupt response—a primary objective of the present invention.

B. Robinson Fails to Disclose or Suggest All Claimed Features

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Robinson fails to satisfy this rigorous standard.

1. Robinson Does Not Teach Interrupting Execution of Operations After Completion of Execution of Any Currently Executing Operation

Claim 1 recites in element (iii) an "an interrupt handler responsive to an interrupt signal to interrupt execution of operations corresponding to instructions of said first instruction set after completion of execution of **any** currently executing operation." Claim 16 recites in (iii) an analogous method step feature: "in response to an interrupt signal, interrupting execution of operations corresponding to instructions of said first instruction set after completion of execution of **any** currently executing operation." Robinson does not disclose this feature.

In the final rejection statement, the Examiner relies on col. 7, line 61-col. 8, line2. In the preceding text in column 7, lines 15-37, Robinson discloses that when a non-native X instruction is translated to a native Y instruction, the translated code instructions are grouped and ordered as follows:

1) a first group of G1 of instructions that get inputs and place those inputs in temporary storage;
2) a second group G2 of instructions that operate on inputs generate modified results that store those results to temporary storage;
3) a third group G3 of instructions that update the memory or register (i.e. update the untranslated instruction, X, state) and are subject to possible exceptions;
4) a last group G4 of instructions that update the memory or register (i.e., update the untranslated instruction, X, state) and are free of possible exceptions (e.g., simple register moves-see Robinson, column 11, lines 53-56).

Then starting at line 61, Robinson states that

> "[i]f an asynchronous event occurs after Y instructions in the groups G1 and G2 have been executed and if there are no Y instructions in the group G3, or if the event occurs after execution of all Y instructions included in the group G2 as indicated by as arrow 73 in Fig. 6, the processing of *the asynchronous event can be briefly delayed as the group G4 instructions are executed with foreknowledge that no state exceptions are possible.*"

In contrast to the claim feature (iii) recited in claims 1 and 16, if Robinson's Group 3 or Group 4 instructions are being executed, then they must be completed prior to

interruption. So in the very text relied on by the Examiner, Robinson describes situations in which the claim language "interrupting execution of operations corresponding to instructions of said first instruction set after completion of execution of any currently executing operation" is not met.

Even if there might be some circumstances where Robinson might perform the interrupt immediately after the finishing the currently executing instruction, Robinson teaches other situations where Robinson's system continues on to execute the next instruction despite an earlier asynchronous event. Those other situations violate the claim language which uses the term "**any** currently executing operation."

This claim feature is highly significant from the standpoint of improving interrupt latency as described in the summary and in the specification. Because interrupt latency is controlled by the worst-case situation, it is not sufficient that many or even most instructions can be interrupted rapidly. What is significant is the guaranteed worst-case interrupt response latency. Because of the claimed interrupt execution after completion of execution of **any** currently executing operation, only a single operation needs to complete before interrupt handling can commence. So the worst case interrupt latency is much improved in the present invention as compared to those situations described in Robinson when the interrupt occurs during execution of a Group 3 instruction and a subsequent Group 4 instruction must be completed before the interrupt is serviced.

On the basis of this missing claim element, the rejections based on Robinson are improper and must be reversed.

2. Robinson Does Not Describe For Each Translation Sequence That No Change Is Made To Input Variables Until Execution Of The Sequence's Final Operation

Claim 1 recites in element (v): "said instruction translator is operable to generate a sequence of one or more sets of translator output signals corresponding to instructions of said first instruction set to represent said at least one instruction of said second instruction set, **each** sequence being such that no change is made to said one or more input variables until a final operation within said sequence is executed." An analogous method step (v) is recited in claim 16. This element is missing in Robinson.

In the final rejection statement, the Examiner relies on col. 11, lines 58-65 of Robinson for the claim clause beginning with "**each** sequence ...." Prior to this text, Robinson describes a "translated sequence that is interrupted before the second instruction group completes is forced to restart at the beginning of the sequence." Column 11, lines 49-51. Alternatively, "[a] translated sequence that is interrupted after the second instruction group completes but before the third instruction group completes is forced to complete the third instruction with simple register moves." Column 11, lines 53-57. Then Robinson describes one possible situation with a 1- or 2-byte non-interlocked write on a single processor. In that one case, the "translated sequence includes a state read-modify-write sequence, and the single write is the group 3 instruction." Col. 11, lines 63-65.

But even assuming for the sake of argument that this one sequence meets the language of claim element (v), Robinson still fails to disclose that "no changes are made to the input variables until execution of a final operation in the sequence" for **each**

translated instruction sequence. Indeed, from the explicit groupings and ordering of the instructions disclosed by Robinson in column 7, instructions *other than the last instruction* in some sequences result in changes to the input variables. In particular, the G3 instructions update memory or a register, and hence, make changes to the input variables. Yet, Robinson clearly teaches that G4 instructions follow the G3 instructions in the ordered sequence and "update X state." See col. 7, lines 22-37. The fact that there might be a sequence when this does not occur does not teach that *each and every sequence* does not make any change to the input variables until the final operation in the sequence.

Unlike Robinson, the instant inventors recognized that by generating the translated sequence of instructions in an order so that no change is made to the input variables until a final operation within the sequence is executed, an interrupt can be serviced directly after execution of any instruction that is currently executing when the interrupt is generated and before execution of a subsequent instruction. This is true regardless of the particular point in an execution sequence when the interrupt occurs. As specified in the claims, if the interrupt signal occurred prior to starting execution of the final operation then, following the interrupt, execution is re-started at the first operation in the translated sequence. That restart can be done without compromising the accuracy of the computation results, since it is known that no changes have yet been made to the input variables. Otherwise, if the interrupt signal occurred after the final operation of the sequence started to execute, then the execution re-starts at the next instruction following

the translated sequence. In this way, interrupt latency is improved without compromising the accuracy of the computation results.

Robinson's focus on preserving instruction granularity is at the cost of slower interrupt response. Indeed, interrupt service may be considerably delayed in Robinson pending execution of a sequence of instructions until the next boundary is reached once execution has progressed beyond Group 2 instructions. While this might be beneficial for preserving instruction granularity, it is at the significant expense of greater interrupt latency.

The Examiner makes an obviousness rejection of remaining claims 8-10 and 13-15 based on the combination of Robinson and Dickol et al. But Dickol does not remedy the deficiencies of Robinson noted above. Therefore, the obviousness rejection is improper and should be reversed.

VIII. CONCLUSION

At least two features of the independent claims are not disclosed or suggested by Robinson. Each missing claim feature is an independent ground for reversal. The Board should reverse the outstanding rejections.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: ______________________________
John R. Lastova
Reg. No. 33,149

JRL/kmm
Enclosures
Appendix A - Claims on Appeal

952904

IX. CLAIMS APPENDIX

1. An apparatus for processing data, said apparatus comprising:

(i) a processor core operable to execute operations as specified by instructions of a first instruction set;

(ii) an instruction translator operable to translate instructions of a second instruction set into translator output signals corresponding to instructions of said first instruction set, at least one instruction of said second instruction set specifying an operation to be executed using one or more input variables;

(iii) an interrupt handler responsive to an interrupt signal to interrupt execution of operations corresponding to instructions of said first instruction set after completion of execution of any currently executing operation; and

(iv) restart logic for restarting execution after said interrupt; wherein

(v) said instruction translator is operable to generate a sequence of one or more sets of translator output signals corresponding to instructions of said first instruction set to represent said at least one instruction of said second instruction set, each sequence being such that no change is made to said one or more input variables until a final operation within said sequence is executed; and

(vi) after occurrence of an interrupt during execution of a sequence of operations representing said at least one instruction of said second instruction set:

(a) if said interrupt occurred prior to starting execution of a final operation in said sequence, then said restart logic restarts execution at a first operation in said sequence; and

(b) if said interrupt occurred after starting execution of a final operation in said sequence, then said restart logic restarts execution at a next instruction following said sequence.

2. The apparatus as claimed in claim 1, wherein said translator output signals include signals forming an instruction of said first instruction set.

3. The apparatus as claimed in claim 1, wherein said translator output signals include control signals that control operation of said processor core and said control signals match control signals produced on decoding instructions of said first instruction set.

4. The apparatus as claimed in claim 1, wherein said translator output signals include control signals that control operation of said processor core and specify parameters not specified by control signals produced on decoding instructions of said first instruction set.

5. The apparatus as claimed in claim 1, wherein said restart logic is part of said instruction translator.

6. The apparatus as claimed in claim 1, wherein said restart logic stores a pointer to a restart location within instructions of said second instruction set that are being translated, said pointer being advanced upon execution of said final operation.

7. The apparatus as claimed in claim 6, wherein said pointer is a program counter value pointing to a memory address of a memory location storing an instruction of said second instruction set currently being translated.

8. The apparatus as claimed in claim 1, wherein instructions of said second instruction set specify operations to be executed upon stack operands held in a stack and said input variables include input stack operands.

9. The apparatus as claimed in claim 8, wherein any stack operands removed from said stack by execution of said at least one instruction of said second instruction set are not removed until after execution of said final operation has commenced.

10. The apparatus as claimed in claim 8, wherein any stack operands added to said stack by execution of said at least one instruction of said second instruction are not added until after execution of said final operation has commenced.

11. The apparatus as claimed in claim 1, wherein said input variables include system state variables not specified within said at least one instruction of said second instruction set.

12. The apparatus as claimed in claim 1, wherein said processor has a register bank containing a plurality of registers and instructions of said first instruction set execute operations upon register operands held in said registers.

13. The apparatus as claimed in claim 12, wherein a set of registers within said register bank holds stack operands from a top potion of said stack.

14. The apparatus as claimed in claim 13, wherein said instruction translator has a plurality of mapping states in which different registers within said set of registers hold respective stack operands from different positions within said stack, said instruction translator being operable to move between mapping states when said final operation is executed so as to update said input variables.

15. The apparatus as claimed in claim 1, wherein said instructions of said second instruction set are Java Virtual Machine instructions.

16. A method of processing data, said method comprising the steps of:

(i) executing operations as specified by instructions of a first instruction set;

(ii) translating instructions of a second instruction set into translator output signals corresponding to instructions of said first instruction set, at least one instruction of said second instruction set specifying an operation to be executed using one or more input variables;

(iii) in response to an interrupt signal, interrupting execution of operations corresponding to instructions of said first instruction set after completion of execution of any currently executing operation; and

(iv) restarting execution after said interrupt; wherein

(v) said step of translating generates a sequence of one or more sets of translator output signals corresponding to instructions of said first instruction set to represent said at least one instruction of said second instruction set, each sequence being

such that no change is made to said one or more input variables until a final operation within said sequence is executed; and

(vi) after occurrence of an interrupt during execution of a sequence of operations representing said at least one instruction of said second instruction set:

(a) if said interrupt occurred prior to starting execution of a final operation in said sequence, then restarting execution at a first operation in said sequence; and

(b) if said interrupt occurred after starting execution of a final operation in said sequence, then restarting execution at a next instruction following said sequence.

17. A computer program product including a computer program for controlling a computer to perform the method of claim 16.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.